#8

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/17166 A2

(51) International Patent Classification[7]: H04L 12/00

(21) International Application Number: PCT/US00/23647

(22) International Filing Date: 29 August 2000 (29.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/387,558 1 September 1999 (01.09.1999) US

(71) Applicant: INSILICON [US/US]; 411 East Plumeria Drive, San Jose, CA 95134 (US).

(72) Inventor: KALAPATAPU, Ramana; 3264 Palmo Court, San Jose, CA 95135 (US).

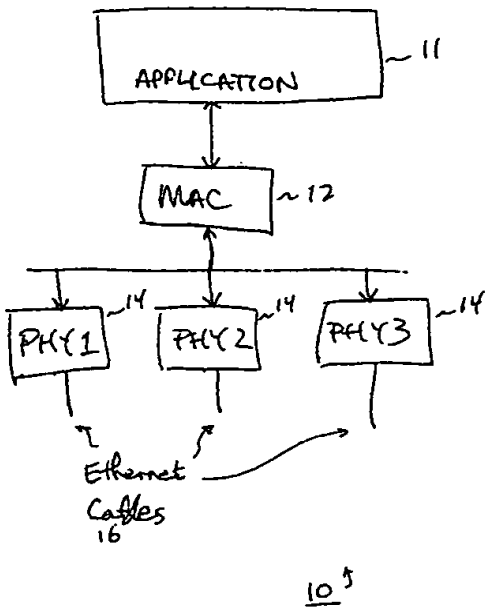
(74) Agents: PATTI, Carmen, B. et al.; Wildman, Harrold, Allen & Dixon, 225 West Wacker Drive, Chicago, IL 60606 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KR (utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: ETHERNET 10/100 MEDIA ACCESS CONTROLLER CORE



(57) Abstract: A design tool for designing a MAC contains the functionality for performing an Ethernet data link layer protocol and also includes a variety of system-level functions. These system level functions include, for example, RMII, SMII, VCI, preamble generation and removal, automatic thirty-two bit CRC generation and checking, insertion and stripping of padding bytes on transmission and reception, address filtering, a configurable counter for system management support, control of MII compatible PHYs, CSMA/CD, a test environment for verifying functional compliance, VLAN support, synchronization of the MII clocks to the application clock, and Big/Little endian data format support for the application interface. The inclusion of the system-level functions simplifies the design of a MAC, reduces the time to market, ensures compatibility with the Ethernet standards, and enables increased certainty of design due to the built-in test environment.

WO 01/17166 A2

IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

ETHERNET 10/100 MEDIA ACCESS CONTROLLER CORE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to the field of digital communication and more particularly to the field of Ethernet medium access controllers.

2. Description of the Related Art

Designing controllers for communication devices can be a complicated process due, in part, to the large variety of functions that must be performed. Standards such as the Open Systems Interconnection model (the "OSI" model), promulgated by the International Standards Organization, provide a convenient and logical way to categorize the many functions involved in digital communication. The OSI model includes seven different layers, and the interfaces which are defined between these layers allow system developers to develop systems in a modular fashion. The seven layers are: (i) the physical layer, which is concerned with the connection to the communications medium, (ii) the data link layer, which is concerned with reliable data transfer, (iii) the network layer, which is concerned with data routing and addressing, (iv) the transport layer, which is concerned with connections between hosts and ensuring data reception, an example of a transport layer protocol being the TCP protocol, (v) the session layer, which is concerned with logical connections between hosts, as well as security, (vi) the presentation layer, which is concerned with file formats and data conversion, and (vii) the application layer, which is concerned with standards for user or application interfaces.

The OSI model has also served as a building block for various standards, such as the IEEE 802.3 Ethernet standards or the IEEE 1394 standards. The OSI model and the many standards built on top of it have a further advantage in that they allow third party developers to create design tools which incorporate much of the required functionality for the various layers and which are compatible with the interfaces defined by the relevant

standards. System designers can then use a function call to implement a function, rather than coding it themselves.

However, the system designer must still create a great deal of code to complete a typical communications system. The present invention is directed to improving this situation.

SUMMARY OF THE INVENTION

This situation is improved with a design tool for designing a medium access controller ("MAC") which contains the functionality for performing an Ethernet data link layer protocol and also includes a variety of system-level functions such as support of additional physical layer interface protocols and application interface protocols, packet handling features, system management support, application interface functions, and a test environment. The inclusion of the system-level functions simplifies the design of a MAC, reduces the time to market, ensures compatibility with the Ethernet standards, and enables increased certainty of design due to the built-in test environment. Additionally, embodiments can be adapted to different standards and different layers of the OSI model, and be based on entirely different models altogether.

Briefly, in accordance with one aspect of the present invention, there is provided a computer program product including computer readable program code for designing a controller. The computer readable program code includes three program parts. A first program part is for implementing an application interface. A second program part is for implementing a physical layer interface. A third program part is for implementing a system-level function.

Briefly, in accordance with another aspect of the present invention, there is provided a computer program product including computer readable program code for designing a controller. The computer readable program code includes two program parts. A first program part is for substantially implementing a medium access control sublayer protocol, and a second program part is for implementing a system-level function.

Briefly, in accordance with another aspect of the present invention, there is provided a method of designing a controller using a software package. The method includes utilizing the software package which includes an application interface functionality, a physical layer interface functionality, and a system-level function functionality. The method further

includes incorporating the application interface functionality, the physical layer interface functionality, and the system-level function functionality into a controller design

Briefly, in accordance with another aspect of the present invention, there is provided a method of designing a controller using a software package. The method includes utilizing the software package which includes a medium access control sublayer functionality and a system-level function functionality. The method further includes incorporating the medium access control sublayer functionality the system-level function functionality into a controller design.

Briefly, in accordance with another aspect of the present invention, there is provided a design tool for designing a controller. The design tool includes a medium access control sublayer emulator which substantially emulates a medium access control sublayer protocol. The design tool also includes a system-level function emulator.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a high-level block diagram of a system which utilizes a MAC.

FIG. 2 shows a high-level block diagram of main components in a MAC.

FIG. 3 shows a high-level block diagram of a system which utilizes a MAC in a cable modem.

FIG. 4A shows a high-level block diagram of a system which utilizes a MAC in a PCI to Ethernet controller.

FIG. 4B shows further details of the PCI to Ethernet controller of FIG. 4A.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

1. System Overview

System design has been growing increasingly complex over the last decade. Due to increased integration densities and system speeds, entire systems are now able to be placed on single application specific integrated circuits ("ASICs") or other devices. The entire design can be done in software, for example Verilog, and then downloaded into a device. The system designer can utilize various design tools to assist in implementing a given standard, such as the IEEE 802.3 Ethernet standard, but must still develop the code for the rest of the system. This situation involves large design times and increased time to market, and requires substantial expertise and experience.

An embodiment of the present invention simplifies the design process and reduces the design time required. This embodiment provides a design tool for designing for an Ethernet medium access controller ("MAC"). The medium access control sublayer is the lower sublayer of the data link layer of the OSI model. The embodiment includes the functionality for implementing the Ethernet layer two protocol and also incorporates functionality for system-level functions. The availability of the system-level functions greatly simplifies the design process for a system designer and reduces the required design time. The embodiment is suited for a variety of design applications, including system-on-chip, switching, routing, and network interface card applications.

FIG. 1 shows a high-level block diagram of a system 10 which utilizes a MAC. The MAC 12 interfaces, on one end, to one or more PHYs 14. A PHY 14 is a physical layer device, and the interface from the MAC 12 to the PHY 14 is a physical layer interface. The physical layer interface of the MAC 12 is preferably the media independent interface ("MII") which defines the interface between the physical layer and the data link layer of the OSI model, and which is specified in the IEEE 802.3 standard.

The MAC 12 also interfaces, on the other end, to an application 11. The application 11 is a generic element identifying the fact that the MAC is only a layer two device and that additional functionality is necessary before interfacing to a user. The application 11 could be a layer three networking device. If the MAC 12 also performs higher level functionality, however, then the application 11 could be a transport layer device, a session layer device, a presentation layer device, an application layer device, or any combination if the OSI model is not strictly followed. The interface between the MAC 12 and the application 11 is called the application interface. Preferably, the application interface of the MAC 12 includes the Virtual Component Interface ("VCI") which was defined by the Virtual Socket Interface ("VSI") Alliance. The VCI contains transaction layer logic and FIFOs to handle data transfers between the application 11 and the MAC 12.

As already stated, the MAC 12 performs more than the layer two Ethernet protocol by including the VCI and various system-level functions which are described further below. The MAC 12 is not limited, therefore, to performing functionality in the OSI model or any particular standard. Preferably however, the MAC 12 performs the entire Ethernet standard as defined in IEEE standards 802.3 (half-duplex), 802.3x (full-duplex), and 802.3u (100 Mbytes/sec or 100 Mbps). Further, the MAC 12 also is preferably tested to

verify compliance with these standards. Other embodiments provide only substantially all of the functionality of the data link layer protocol or the medium access control sublayer protocol. While a MAC 12 according to the present invention also could be designed with less than substantially all of the entire medium access control sublayer functionality, it would be less marketable.

As stated earlier, in one embodiment the MAC 12 is an Ethernet MAC. FIG. 1 illustrates such a case, by showing the PHYs 14 each connected an Ethernet cable 16. However, a suitably designed MAC 12 may be used with other communication standards, for example and without limitation, the 1394 standard.

FIG. 2 shows a high-level block diagram of a MAC 12. A brief description of the MAC 12 and FIG. 2 follows in this paragraph, and a more detailed analysis of the MAC 12 is included in the Appendix. In this embodiment, the MAC 12 contains (i) a MAC Block 45, (ii) a Control Status Register Block ("CSR" Block) 41, (iii) a MAC Host Block 42, (iv) an Address Check Block 43, and (v) a Station Management Block 44. The MAC Block 45 implements the functionality of the Ethernet Medium Access Control sublayer for both transmit and receive operations. The MAC Block 45 contains a RX CRC block 46, a TX CRC block 50, a RX block 47, a TX block 49, a Flow Control block 48, a RX PHY Interface 51, and a TX PHY Interface 52. The CSR Block 41 contains control and status registers for the operation of the MAC 12 and also contains configurable counters, which are described further below. The MAC Host Block 42 handles the synchronization of control and data signals across the application interface (see FIG. 1). The Address Check Block 43 checks the destination address field of all the incoming packets. The Station Management Block 44 controls the read/write transactions to PHY registers which are located in a PHY.

2. Examples

Two examples of products which use an Ethernet MAC are described below. FIG. 3 shows the first, which is a PCI to Ethernet controller. FIGS. 4A-4B show the second, which is an Ethernet to Cable modem.

FIG. 3 shows a high-level block diagram of a system 20 which utilizes a MAC 12 in a cable modem 21 which connects to both an Ethernet cable 16 and a coaxial cable 23. The cable modem 21 also includes the PHY 14 to connect to the Ethernet cable 16, and cable

modem functionality 25 to perform all of the necessary functions on the other side of the MAC 12. FIG. 3 also shows a PC 27 and a host 28 connected to the Ethernet cable 16.

FIG. 4A shows a high-level block diagram of a system 30 which utilizes a MAC 12 in a PCI to Ethernet controller 35. The PCI to Ethernet controller 35 connects to a motherboard 31 and both are shown as being inside of the PC 27 (see FIG. 3). The PCI to Ethernet controller 35 includes PCI functionality 33, as well as the MAC 12 and the PHY 14. FIG. 4B shows further detail, and illustrates the fact that the PCI Functionality 33 includes a PCI Core 39 and an Additional PCI Functionality 37. FIG. 4B highlights the communication between the MAC 12 and the PCI Core 39, which preferably is a VCI. As described in more detail below, the provision of VCI capability is a system-level function.

3. System-Level Functions

The MAC 12 preferably includes a variety of system-level functions which simplify the process of designing a system on a chip. Whereas the MAC Block 45 (see FIG. 2) implements the basic functions of the Medium Access Control sublayer, many of the system-level functions are provided by the CSR Block 41, the MAC Host Block 42, the Address Check Block 43, and the Station Management Block 44. These system-level functions can include: (i) a reduced MII interface ("RMII"), (ii) a serial MII interface ("SMII"), (iii) a VCI, (iv) preamble generation and removal, (v) automatic thirty-two bit CRC generation and checking, (vi) insertion and stripping of padding bytes on transmission and reception, (vii) address filtering, (viii) a configurable counter for system management support, (ix) control of MII compatible PHYs, (x) Carrier Sense Multiple Access Collision Detection ("CSMA/CD"), (xi) a test environment for verifying functional compliance of a system design with a specified standard, (xii) Virtual LAN ("VLAN") support, (xiii) synchronization of the MII clock(s) to the Application clock, and (xiv) Big/Little endian data format support for the application interface. Each of these system-level functions is further described below.

As indicated, a variety of interfaces are preferably supported. Both RMII and SMII lower the pin count required to implement the interface between the physical layer and the data link layer. A lower pin count can be very useful in switch applications that integrate multiple PHYs. The VCI was described earlier.

Several of the system-level functions deal with packet handling. These include preamble generation and removal, automatic or manual thirty-two bit CRC generation and checking, and insertion and stripping of padding bytes on transmission and reception. Further, overhead is minimized with the flexible address scheme which filters out data in the network that is not addressed to the node in which the MAC resides. Complete status for both transmit and receive packets is also preferably available.

Several additional system-level functions deal with system management support. The configurable counters can be used to monitor the system by tracking various events such as the number of CRC errors, the number of network collisions, the number of runt frames, etc. Another system management support function is the control of MII compatible PHYs. For example, this feature can force PHYs to run at 10 Mbps or 100 Mbps, and can configure them to run in full-duplex mode or half-duplex mode. The MAC also preferably has the additional system management support function of being able to shut down individual PHY ports, which can be a useful function in switch applications.

The CSMA/CD protocol is used to control access to a communications medium by multiple devices and to handle collisions. In half-duplex mode, collision detection and auto-retransmission on collisions is preferably automatically handled. In full-duplex mode, flow control and automatic generation of control frames is also handled.

Several of the functions relate to the application interface. This includes VLAN support. Additionally, an internal synchronization block is preferably used to synchronize the signals from the MII transmit and receive clocks to the application clock provided by the application. Further, the Big/Little endian data format support is for a data bus on an application bus, which is further described in the Appendix.

In the test environment, the design can be tested for functional compliance with a given standard. Random test vectors can also be generated to test the actual design application output. As an example of a test feature, an external or internal loop back capability is preferably provided for the MII interface.

4. Applications and Variations

The MAC embodiment of the present invention provides a design tool which can implement and/or emulate an application interface, a physical layer interface, and a system-level function. The design tool allows this functionality to be incorporated into the design

of a controller by using an advanced set of libraries which are invoked with function calls, but it can make this capability available in other ways as well. Preferably the tool can be used to implement, or substantially implement, a medium access control sublayer protocol, a data link layer protocol, and/or a link layer core (also referred to as a data link layer core).

In accordance with the present invention, the functionality disclosed in this application can be, at least partially, implemented by hardware, software, or a combination of both. This may be done, for example, with a computer system running Verilog, or other design programs. Moreover, this functionality may be embodied in computer readable media or computer program products to be used in programming an information-processing apparatus to perform in accordance with the invention. Such media or products may include magnetic, magnetic-optical, optical, and other types of media, including for example 3.5 inch diskettes. This functionality may also be embodied in computer readable media such as a transmitted waveform to be used in transmitting the information or functionality.

Additionally, software implementations can be written in any suitable language, including without limitation high-level programming languages including high-level hardware description languages ("HDLs") such as Verilog or VHDL, mid-level and low-level languages, assembly languages, and application-specific or device-specific languages. Such software can run on a general purpose computer such as a 486 or a Pentium, an application specific piece of hardware, or other suitable device.

In addition to using discrete hardware components in a logic circuit, the required logic may also be performed by an application specific integrated circuit ("ASIC") or other device. The technique may use analog circuitry, digital circuitry, or a combination of both. Embodiments may also include various hardware components which are well known in the art, such as connectors, cables, and the like.

The principles, preferred embodiments, and modes of operation of the present invention have been described in the foregoing specification. The invention is not to be construed as limited to the particular forms disclosed, because these are regarded as illustrative rather than restrictive. Moreover, variations and changes may be made by those of ordinary skill in the art without departing from the spirit and scope of the invention.

APPENDIX

1 Introduction

1.1 General Description

The Ethernet Media Access Controller (MAC110) core incorporates the essential protocol requirements for operation of an Ethernet/IEEE 802.3 compliant node, and provides interface between the host subsystem and the Media Independent Interface (MII). The MAC110 core can operate either both in 100Mbps mode or the 10Mbps mode based on the clock provided on the MII interface (25/2.5 MHz).

The MAC110 Core operates both in half-duplex mode and full-duplex modes. When operating in the half-duplex mode, the MAC110 Core is fully compliant to Section 4 of ISO/IEC 8802-3 (ANSI/IEEE Standard) and ANSI/IEEE 802.3. When operating in the full-duplex mode, the MAC110 core is compliant to the IEEE 802.3x standard for full-duplex operations.

The MAC110 Core provides programmable enhanced features designed to minimize host supervision, bus utilization, and pre- or post-message processing. These features include ability to disable retries after a collision, dynamic FCS generation on a frame-by-frame basis, automatic pad field insertion and deletion to enforce minimum frame size attributes, automatic retransmission and detection of collision frames.

The MAC110 core can sustain transmission or reception of minimal-sized back-to-back packets at full line speed with an interpacket gap (IPG) of 90.6 us for 10-Mb/s and 0.96 us for 100-Mb/s.

The five primary attributes of the MAC block are:

- Transmit and receive message data encapsulation
  - Framing (frame boundary delimitation, frame synchronization)
  - Error detection (physical medium transmission errors)
- Media access management
  - Medium allocation (collision avoidance, except in full-duplex operation)
  - Contention resolution (collision handling, except in full-duplex operation)
- Flow Control during Full Duplex mode
  - Decoding of Control frames(PAUSE Command) and disabling the transmitter
  - Generation of Control Frames.
- Serial Interface Control
  - Support of MII protocol to interface with a MII based PHY.
  - Optional support of RMII protocol on the Ethernet side.
  - Support of standard ENDEC interface to interface with a 10BASE PHY.
- Management Interface support on MII
  - Generation of Management frames on the MDC/MDIO pins to talk to an external frame.

1.2 MAC110 Features

The MAC110 core has the following features:

- Compliant with IEEE 802.3, 802.3u, 802.3x Specification
- Supports 10/100 Mb/s data transfer rates.
- IEEE 802.3 compliant MII interface to talk to an external PHY.
- VLAN support.
- Supports both full-duplex/half-duplex operations.
- Support of CSMA/CD Protocol for half-duplex.
- Supports flow-control for full-duplex operation.
- Collision detection and auto retransmission on collisions in half-duplex mode.
- Management support by using variety of counters.
- Preamble generation and removal.
- Automatic 32 bit CRC generation and checking.
- Options to insert PAD/CRC32 on transmit.
- Options to Automatic Pad stripping on the receive packets.
- Provides External and internal loop back capability on the MII Interface.
- Contains a variety of flexible address filtering modes on the Ethernet side:
  - One 48 bit Perfect address
  - 64 hash-filtered multicast addresses
  - Pass all multicast addresses
  - Promiscuous Mode
  - Pass all incoming packets with a status report.
- Separate 32-bit status returned for transmit and receive packets.
- VCI Compliant Application Interface Bus.
- Separate 32-bit Receive, Transmit and Host Interfaces on the Application Bus.
- Internal synchronization block to synchronize the signals from MII Receive/Transmit clocks to the Application Clock provided by the Application.
- Big/Little endian data format support for data bus on the Application Bus.

2 Hardware Overview

2.1 Block Diagram

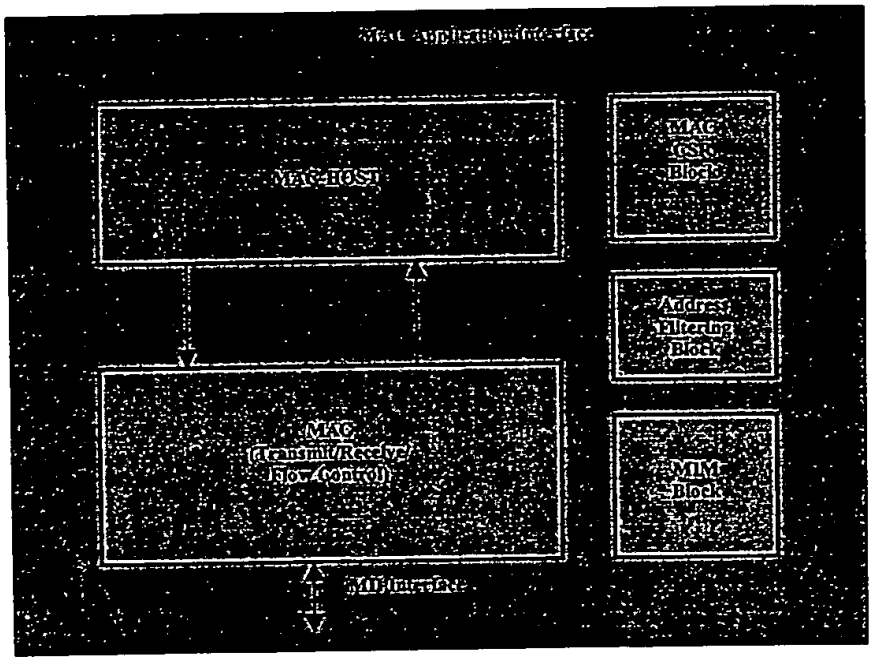


Figure 2-1: MAC110 Block Diagram

2.2 *Block Overview*

The following list describes the *MAC110 Core's* hardware components, which are shown in Figure 2-1. Each of these blocks are defined in detail in the later chapters.

- **10/100-Mb/s MAC Block (MAC):** This block handles all the functionality of the Ethernet MAC Layer for both transmit and receive operations. CSMA/CD protocol is being implemented for half-duplex and the flow control for the full duplex.
- **MAC CSR (MCS) Block:** This block contains the control and status registers for the operation of the MAC110 Core. This block also contains the Ethernet RMON counters. The registers/counters in this block can be accessed through the Host Interface on the Application bus.
- **MAC-Host(MHT) Block:** This block handles the synchronization of control and data signals between the Host bus and the MAC Interface. This block also does the byte packing/unpacking and byte swapping to handle big/little endian data formats.
- **Address Check (ACH) Block:** This block checks the destination address field of all the incoming packets. Based on the type of address decoding selected this will indicate the MAC RX Interface block the result of the Address checking.
- **MII Management Block (MIM):** This block controls the read/write transactions to the PHY Registers which are in the external MII based PHY Controller ASIC, through the MII Management Interface using MDC and MDIO signals.

2.3 *Interfaces*

The following list describes the *MAC110 Core* interfaces:

- **MII Interface:** The *MAC110* Core interfaces to the Ethernet cable with the IEEE defined MII (Media Independent Interface) interface. This is a 4-bit parallel interface. Any of the standard Ethernet PHY Controller chips can be hooked on to this interface for connecting to the Ethernet cable. This interface is specified in the IEEE 802.3u specification. This is the default interface on the Ethernet side.
- **ENDEC Interface:** The *MAC110* Core also interfaces to the Ethernet cable with the industry standard Encoder/Decoder (ENDEC) interface, when this port is selected using the *PortSelect* bit in the MAC Control register. This is 7-wire serial interface that operates at 10 MHz (for 10Mbps operation only). This interface can be used to connect to the external 10BASE-2 PHY chips.
- **RMII Interface:** The *MAC110 Core* also interfaces to the Ethernet PHY chip through an optional RMII Interface. When operating with this interface, the *MAC110 Core* uses only 2-bit for data on each direction. The detailed protocol is specified in the RMII specification.
- **MII Management Interface:** The MII Provides a two-wire management interface so that the *MAC110 Core* can control and receive status from external PHY devices.
- **Application Bus Interface:** The MAC110 Core interfaces to the Application logic though the Application Bus Interface. This interface is subdivided into three separate Interfaces. Each of these interfaces follow the same VCI compliant bus protocol. The data bus on these interfaces is 32-bit wide.
  - *Transmit Interface*: This interface is used to transmit the frames from the Application to the Ethernet.
  - *Receive Interface*: This Interface is used to receive the frames from the Ethernet to the Application.
  - *Host Interface*: This interface is used to access the registers/counters in the CSR block.

3 Signal Descriptions

3.1 Pinout

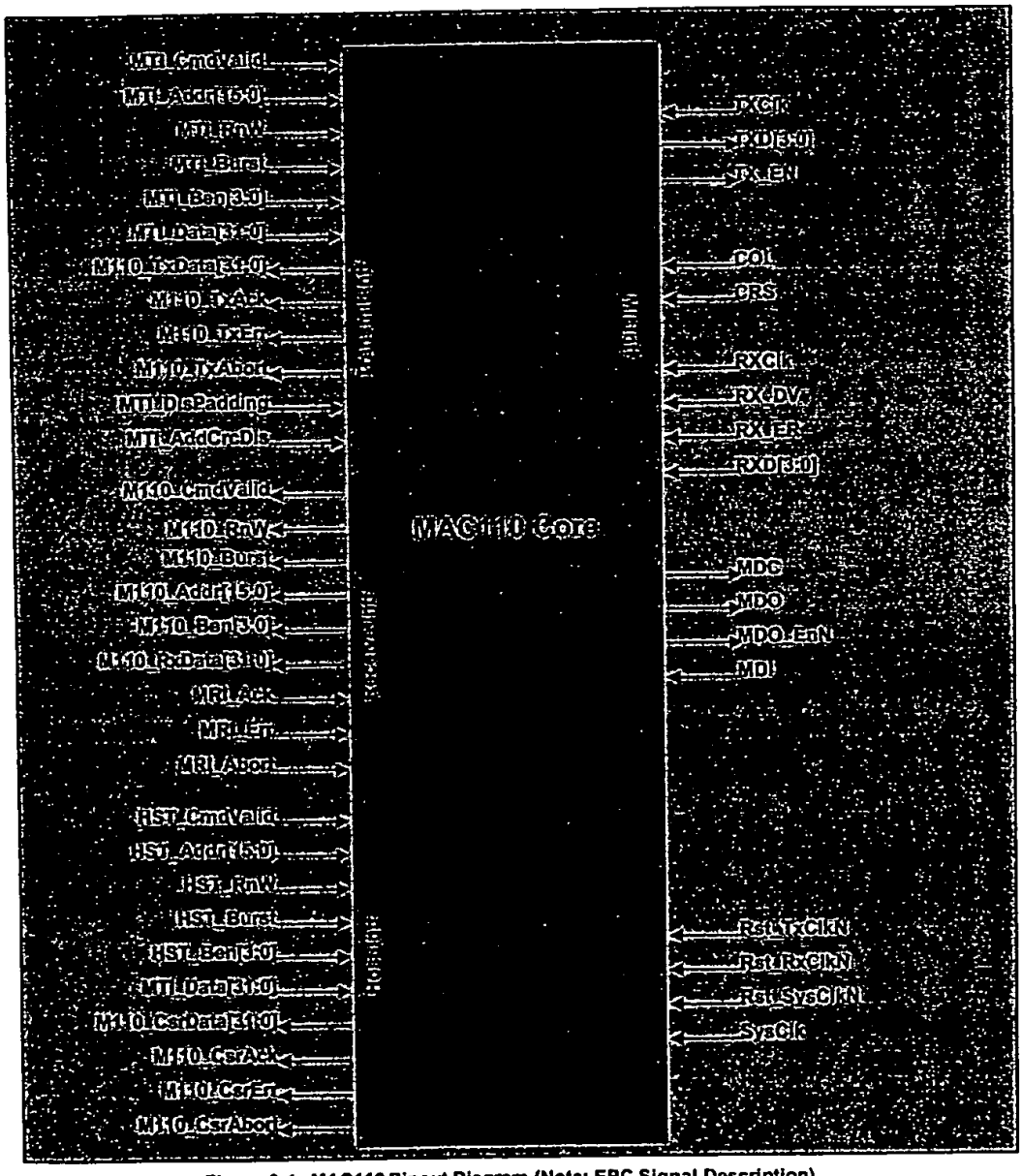


Figure 3-1: *MAC110* Pinout Diagram (Note: EPC Signal Description)

3.2 Signal Description

The MAC110 Core needs three clock inputs. The SysClk from the Application and the MII TX_Clk and RX_Clk on the MII Interface. The signals that are output from the MAC110 Core to the Application are prefixed with "**M110_**". The signals that are input from the Application on the Transmit Interface are prefixed with "**MTI_**" (MAC Transmit Interface). Similarly the signals that are on the Receive Interface are prefixed with "**MRI_**" (MAC Receive Interface), and on the Host Interface signals are prefixed with "**HST_**". All control signals that end with 'N' are low asserted signals, i.e., when asserted they are at logic '0' and when de-asserted they are at logic '1'.

Signal Name	Direction	Description
1. Clock and Reset Signals		
SysClk	IN	This is the free running system clock provided by the Application. The signals on the Application bus are synchronous to this clock. The Application Interface logic, and the registers/counters in the MAC110 Core runs on this clock. The Transmit and Receive state machines run on their respective clocks. The MAC-HOST block in the MAC110 Core synchronizes the signals between the SysClk to the Transmit/Receive clocks respectively.
Rst_SysClkN	IN	This is the Reset signal synchronous to the SysClk. All flip-flops/registers/counters inside the MAC110 Core that runs on the SysClk goes to the default state when Rst_SysClkN is asserted. This is an active low signal.
Rst_TxClkN	IN	This is the Reset signal synchronous to the MII TX_CLK. All flip-flops inside the MAC block that are part of transmit path which runs on the TX_CLK goes to default state when Rst_TxClkN is asserted. This is an active low signal.
Rst_RxClkN	IN	This is the Reset signal synchronous to the MII RX_CLK. All flip-flops inside the MAC block that are part of receive path which runs on the RX_CLK goes to default state when Rst_RxClkN is asserted. This is an active low signal.
2. MII Interface Signals		
TXClk	IN	*TXClk* (Transmit Clock) is a continuous clock that provides for the timing reference for the transfer of the *TX_EN*, *TX_ER*, and *TXD* signals from the MAC110 Core to the Ethernet PHY Controller. *TXClk* is sourced by the Ethernet PHY Controller chip. The operating frequency of the *TXClk* is 25 MHz when operating at 100-Mb/s and 2.5 MHz when operating at 10-Mb/s.
TX_EN	OUT	*TX_EN* indicates that the MAC110 Core is presenting nibbles on the MII for transmission. It will be asserted by the Mac110 Core with the first nibble of the preamble and will remain asserted while all nibbles to be transmitted are presented to the MII. *TX_EN* will be negated prior to the first *TXClk* following the final nibble of the frame. *TX_EN* is driven by the MAC110 Core and will transition synchronously with respect to the *TXClk*. When asserted the TX_EN will be at logic '1' and it will be at logic '0' while de-asserted.
TXD[3:0]	OUT	*TXD* is a bundle of 4 data signals *TXD[3:0]* that are driven by the MAC110 Core. TXD[3:0] will transition synchronously with respect to the *TXClk*. For each *TXClk* period in which *TX_EN* is asserted, *TXD[3:0]* will have the data to be accepted by the Ethernet PHY Controller chip. *TXD[0]* is the least significant bit. While *TX_EN* is de-asserted the data presented on *TXD[3:0]* should be ignored.
RXClk	IN	*RXClk* is a continuous clock that provides the timing reference for the transfer of the RX_DV, RX_ER, and *RXD* signals from the Ethernet PHY Controller to the MAC110 Core. *RXClk* is sourced by the Ethernet PHY Controller chip. The *RXClk* shall have a frequency equal to 25 % of the data rate of the received signal on the Ethernet Cable.
RX_DV	IN	*RX_DV* (Receive Data Valid) is driven by the external Ethernet PHY Controller to indicate the MAC110 Core that it is presenting the recovered

Signal Name	Direction	Description
		and decoded nibbles on the *RXD[3:0]* bundle and that the data on *RXD[3:0]* is synchronous to *RXClk*. RX_*DV* shall transition synchronously with respect to the *RXClk*. RX_*DV* shall remain be asserted continuously from the first recovered nibble of the frame through the final recovered nibble, and shall be negated prior to the first *RXClk* that follows the final nibble. When asserted the RX_*DV* will be at logic '1' and it will be at logic '0' while de-asserted.
RX_ER	IN	RX_ER (Receive Error) is driven by the Ethernet PHY Controller chip. RX_ER shall be asserted for one or more RXClk periods to indicate to the *MAC110 Core* that an error (e.g., a coding error, or any error that the PHY is capable of detecting, and that otherwise be undetectable by the MAC) was detected some where in the frame presently being transferred from the PHY to the MAC110 Core. RX_ER shall transition synchronously with respect to RXClk. While RX_DV is de-asserted, RX_ER will have no effect on the MAC110 Core. When asserted the *RX_ER* will be at logic '1' and it will be at logic '0' while de-asserted.
RXD[3:0]	IN	RXD is a bundle of four data signals RXD[3:0] that transition synchronously with respect to the RXClk. RXD[3:0] are driven by the Ethernet PHY Controller chip. For each RXClk period in which RX_DV is asserted, RXD[3:0] transfer four bits of recovered data from the PHY to the MAC110 Core. RXD[0] is the least significant bit. While RX_DV is de-asserted, RXD[3:0] will have no effect on the MAC110 Core.
CRS	IN	CRS shall be asserted by the Ethernet PHY Controller Chip when either transmit or receive medium is non idle. CRS shall be de-asserted by the PHY when both the transmit and receive medium are idle. The PHY shall ensure that CRS remains asserted throughout the duration of a collision condition. The transitions on the CRS signal are not synchronous to either the TXClk or the RXClk.
COL	IN	COL shall be asserted by the Ethernet PHY Controller chip upon detection of a collision on the medium, and shall remain asserted while the collision condition persists. The transitions on the COL signal are not synchronous to either the TXClk or the RXClk. The COL signal is ignored by the MAC110 Core when operating in the full-duplex mode.
MII_MDC	OUT	MDC is sourced by the MAC110 Core to the Ethernet PHY Controller as the timing reference for transfer of information on the MDI/MDO signals. MDC is an aperiodic signal that has no maximum high or low times. The minimum high and low times for MDC will be 160 ns each, and the minimum period for MDC will be 400 ns, regardless of the nominal period of TXClk and RXClk.
MDI	IN	MDI is the data input signal from the Ethernet PHY Controller. The Read Data is driven by the PHY synchronously with respect to the MDC clock during the read cycles.
MDO	OUT	MDO is the data output signal from the MAC110 Core that is used to drive the control information during the Read/Write cycles to the External PHY Controller. The MDO signal is driven by the MAC110 Core synchronously with respect to the MDC.
MDO_EnN	OUT	MDO_EnN is the tristate enable signal to drive the MDO on to the MDIO pin. This is a low asserted signal.
3. MAC110 Transmit Interface Signals		

Signal Name	Direction	Description
MTI_CmdValid	IN	MTI_CmdValid (Command Valid) is asserted by the Application's transmit Interface to indicate the start of transaction and is asserted through out the transaction. The Address bus, Read/Write signal, Burst Signal should be valid when the MTI_CmdValid is asserted.
MTI_Addr[15:0]	IN	MTI_Addr (Address) is the address for the current data transfer. The MTI_Addr is valid when the MTI_CmdValid is asserted. The MAC110 Core uses only four word aligned addresses and all other addresses are reserved. The following explains the addresses used by the MAC110 Core for Transmit cycles.. MTI_Addr - Meaning 16'h1000 - First word (or part of the word) of the new frame 16'h1004 - 2nd to last but one word (or part of the word) of the frame 16'h1008 - Last word (or part of the word) of the frame. 16'h100C - Address of the Transmit Status. Others - Reserved
MTI_RnW	IN	MTI_RnW (Read/Write) indicates whether the current transaction is a read or a write transaction. When asserted high, it is a read transaction and when asserted low, it is a write transaction. The MTI_RnW is valid when the MTI_CmdValid is asserted.
MTI_Data[31:0]	IN	MTI_Data[31:0] (Data) is the write data to be used for this transaction. In case of a burst transaction, the data is updated on every clock M110_TxAck strobe is asserted. The byte-enables MTI_Ben indicate valid bytes on the MTI_Data bus for the current word transfer.
MTI_Ben[3:0]	IN	MTI_Ben[3:0] (Byte Enables) is used to indicate the valid bytes in the 32 bit data. When the bit is set, it indicates the corresponding data byte is valid. All the MTI_Ben[3:0] signals should be asserted for read transactions.
MTI_Burst	IN	MTI_Burst (Burst Transfer) is used to indicate that the current transaction is a burst transaction. The MTI_Burst is asserted along with the MTI_CmdValid and is deasserted before the last data transfer. The MAC110 Core (slave) can stop the burst transaction by asserting M110_TxErr or M110_TxAbort signal. If the MTI_Burst is not asserted along with the MTI_CmdValid signal, it indicates that the current transaction is a single data transfer transaction.
M110_TxAck	OUT	M110_TxAck (Data Ack) is asserted by the MAC110 Core to indicate the completion of data transfer. In case of the write transaction, the data presented on the MTI_Data bus is accepted and in case of read transaction, the data is available on the M110_TxData bus. In case of Burst transaction, the M110_TxAck signal is asserted for every data transfer and non-assertion of M110_TxAck signal while MTI_CmdValid is asserted is an indication of a wait state.
M110_TxErr	OUT	M110_TxErr (Error/Stop) is asserted by the MAC110 Core either to stop the current burst transfer (with/without) data transfer or to indicate the Application that the MAC110 Core cannot complete the current transfer at this time, and should retry at a later time (retry). When the Buffers (FIFO) in the MAC110 Core is full, the MAC110 Core will assert M110_Err signal to retry a new transaction until the buffers are not full.
M110_TxAbort	OUT	M110_TxAbort (Abort) is asserted by the MAC110 Core to indicate that the current frame is being aborted by the MAC110 Core. The frame is aborted because of any of the following conditions: Underrun condition Late Collision Too many Retries Loss of Carrier No Carrier Excessive Deferral Jabber TimeOut The Application should stop the current transaction and go for Status Read on an abort condition. The MAC110 Core will never assert the Abort signal for the First Word write or the Status Read.

Signal Name	Direction	Description
M110_TxData[31:0]	OUT	M110_TxData[31:0] (Read Data) is the data bus that is used for the read transactions. The Application does a read transaction to read the Transmit Status on the completion of the current frame transmission or on an error condition. The M110_TxData is valid when the M110_TxAck is asserted for the read transcation.
MTI_DisPadding	IN	*(Additional per frame control signal)* MTI_DisPadding (Disable Padding) controls the behavior of the MAC110 Core's Automatic Padding feature for the frames less than 64 bytes (data length 0 to 46 bytes). This can be changed for every new frame and when asserted the padding feature is disabled. The MHT block inside the MAC110 core synchronizes the signal to the MII_TxClk and passes to the MAC block. If the MTI_DisPadding is asserted, the MAC110 Core will not pad to the out-going frame even if the frame is less than 64 bytes. This will create a minFrameSize violation on the Ethernet. If MTI_DisPadding is deasserted, the MAC will automatically pad the outgoing frames to meet the minFrameSize requirement and CRC is automatically added if padded. This signal is valid through out the frame's transmission sequence (from First Word write to status read), and should not be toggled during transmission sequence.
MTI_AddCrcDis	IN	*(Additional per frame control signal)* MTI_AddCrcDis (Add CRC disable) indicates the MAC110 Core whether to add the FCS field or not, to the current out-going frame. When asserted, the CRC field is not appended to the packet. The MHT block inside the MAC110 Core synchonizes the signal to the MII_TxClk and passes it to the MAC block. This signal is valid through out the frame's transmission sequence (from First Word write to status read), and should not be toggled during transmission sequence.
4. MAC110 Receive Interface Signals		
M110_CmdValid	OUT	M110_CmdValid (Command Valid) is asserted by the MAC110 Core to do a new transaction on the MAC110 Receive Interface. The M110_CmdValid is asserted through out the transaction. The address indicates the current phase of the transaction.
M110_Addr[15:0]	OUT	M110_Addr[15:0] (Address) is the address for the current transaction. The M110_Addr is valid when the M110_CmdValid is asserted. The MAC110 Core uses only four word aligned addresses for the Receive transactions and all other addresses are reserved. The following explains the addresses used by the M110 Core for receive transactions. M110_Addr - Meaning 16'h2000 - First word of the new frame 16'h2004 - 2nd to last but one word of the frame 16'h2008 - Last word(or part of word) of the frame. 16'h200C - Receive Status. Others - Reserved
M110_RxData[31:0]	OUT	M110_RxData[31:0] (Data) is the write data to be used for the current data transfer. In case of a burst transaction, the data is updated on every clock MRI_Ack strobe is asserted. The byte-enables (M110_Ben) indicate the valid byte lanes in the data bus.
M110_Ben[3:0]	OUT	M110_Ben[3:0] (Byte Enables) indicates the valid byte lines in the 32 bit data. When the bit is set, it indicates the corresponding data byte is valid. The Byte Enables can be toggled only for the last data write transaction, if the number of bytes to be transferred is less than 4. In all other cases the M110_Ben will all be asserted. The M110_Ben[3:0] will be asserted for status write transactions.
M110_RnW	OUT	M110_RnW (Read/Write) indicates whether the current transaction is a read

Signal Name	Direction	Description
		or a write transaction. When asserted high, it is a read transaction and when asserted low, it is a write transaction. The M110_RnW is valid when the M110_CmdValid is asserted. For Receive Interface, only write transactions are used by the M110 Core.
M110_Burst	OUT	M110_Burst (Burst Transfer) is used to indicate that the current transaction is a burst transaction. The M110_Burst is asserted along with the M110_CmdValid and is deasserted before the last data transfer. The Application (slave) can stop the burst transaction by asserting MRI_Err signal. If the M110_Burst is not asserted along with the M110_CmdValid signal, it indicates that the current transaction is a single data transfer transaction.
MRI_Ack	IN	MRI_Ack (Data Ack) is asserted by the Application to indicate the completion of data transfer and the data presented on the M110_RxData bus is accepted. In case of Burst transaction, the MRI_Ack signal is asserted for every data transfer and non-assertion of MRI_Ack signal while M110_CmdValid is asserted is an indication of a wait state. The MRI Interface should not put no more than 4 clocks of wait states before asserting the MRI_Ack. If the number of wait states exceed 4 clocks, the MAC110 Core block can flush the incoming packet and completes the transmission of the received frame by writing the status word.
MRI_Err	IN	MRI_Err (Error/Stop) is asserted by the Application either to stop the current burst transfer (with/without) data transfer or to indicate the MAC110 Core that the Application cannot complete the current transfer at this time, and should retry at a later time (retry). The MRI_Err should ideally be asserted only to stop the current burst transfer. If the MRI_Err is asserted for normal single transactions, the MAC110 Core can flush the incoming packet and complete the transmission of the received frame by writing the status word. The MRI_Err signal can not be asserted for the Status Write from MAC110 Core.
MRI_Abort	IN	MRI_Abort (Abort) is asserted by the Application to Abort the current data transfer because of some unrecoverable error condition. The MAC110 Core at this point will flush the current frame's data from the FIFO and sets the Missed Frame bit in the status and completes the frame transfer by doing a Status Write to the Application.
5. MAC110 Host Interface Signals		
HST_CmdValid	IN	HST_CmdValid (Command Valid) is asserted by the Application to do a Host transaction for accessing the MAC110's CSR Block. The assertion of the HST_CmdValid indicates the MAC110 Core that the address, read/write and in case of write the data is valid. The assertion of this signal is the start of transaction.
HST_Addr[15:0]	IN	HST_Addr[31:0] (Register/Counter Address) indicates the address of the register/counter the Application is accessing in the current Host transaction. This is valid when the HST_CmdValid is asserted. All addresses to the MCS block should be 32 bit aligned. The address supported on the Host Interface starts from 0x0000 to 0xFFFF and the MAC110 Core will ignore transactions to those addresses that are not supported and completes the transaction in normal fashion.
HST_RnW	IN	HST_RnW (Read Write) indicates whether the current transaction is a read or write transaction. HST_RnW is valid when the HST_CmdValid is asserted. When asserted to high (logic '1'), it indicates that the current transaction is a read transaction else it is a write transaction.
HST_Ben[3:0]	IN	HST_Ben (Byte Enables) indicates the bytes that are valid on the 32 bit data bus. This is valid when the HST_CmdValid is asserted. For Host transactions, all the byte enables should be active.
HST_Data[31:0]	IN	HST_Data[3:0] (Data) is the write data that is to be written in to the addressed register, in the current transaction. This is valid when the current

Signal Name	Direction	Description
		transaction is a write transaction and when the HST_CmdValid is asserted.
M110_CsrAck	OUT	M110_CsrAck (Data Ack.) is the data acknowledgement asserted by the MAC110 Core indicating the completion of data transfer that accesses the MAC110's CSR block. This is asserted by the MAC110 Core for one clock. The Application has to de-assert the HST_CmdValid signal when the MAC110 Core block asserts the M110_Ack and thus completing the transaction during the single transfer cycles, else in case of burst transfer cycles, the Application has to place the next data on the HST_Data bus.
M110_CsrErr	OUT	M110_CsrErr (Error/Stop) is asserted by the MAC110 Core to stop the current burst transfer with data transfer(along with M110_CsrAck). The MAC110 Core will never issue retry for the Host transactions (assertion of M110_CsrErr alone).
M110_CsrAbort	OUT	M110_CsrAbort (Abort) is never asserted by MAC110 Core for Host transactions.
M110_CsrData[31:0]	OUT	M110_CsrData[31:0] (Data) is the read data provided by the MAC110 Core for Host transactions. This is the data from the addressed register and is valid when the MAC110_CsrAck is asserted for read transactions only. When the read transaction is addressed to illegal or unsupported register, the MAC110 completes the transaction normally and returns all zero's on the M110_CsrData bus.

Table 3-1: *MAC110* Pinout Description

4 Application Bus Protocol for MAC110 Core

The Application Bus for the MAC110 core is divided into three different interfaces, a Transmit interface, a Receive interface, and finally the Host bus interface to access MAC110 CSR registers. The protocol on all the three interfaces is the (VSI's) VCI compliant protocol. Each of these three interfaces is independent of one another, and the transactions on one interface are not affected by the transactions on the other interfaces. The data bus width on each of these interfaces is 32-bit wide. Each of these interfaces contains a separate 16-bit address bus that is used for identifying the different transactions.

4.1 Transmit Interface Transaction

On the transmit interface, the transaction is always initiated by the Application. The Application initiates the transaction by asserting the MTI_CmdValid signal and placing the transaction address on the MTI_Addr[15:0] bus. The MTI_RnW indicates whether the transaction is a read or write transaction. If the application wants to do a burst transaction, the MTI_Burst should be asserted along with the MTI_CmdValid signal. In case of a write transaction, the data is placed on the MTI_Data[31:0] bus and the byte enables (MTI_Ben[3:0]) indicate the valid byte lines on the MTI_Data bus. In response to this, the MAC110 Core will assert either M110_TxAck indicating a successful completion of the transfer, or asserts M110_TxErr signal indicating the Application to retry the transaction at a later time. In case of a read transaction, the MAC110 core places the read data on the M110_TxData bus along with the M110_TxAck. A detailed discussion on the transmit protocol is discussed later in the chapter. The following sub-sections show timing diagrams to illustrate the various transactions on the Transmit Interface.

4.1.1 Single Write Transaction with Normal Completion

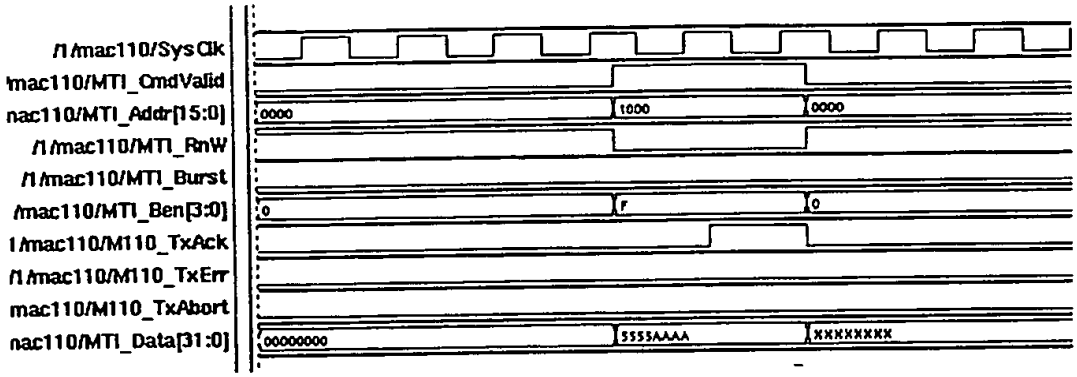


Figure 4-1: Single Write Transaction with normal completion.

Figure 4-1 shows a timing diagram in which the Application wants to do a single write transaction to address 16'h1000 (First word write in the transmit frame sequence). The Application asserts the MTI_CmdVallid signal and places the address on the MTI_Addr bus. The MTI_RnW is asserted low, indicating it as a write transaction, the MTI_Burst is deasserted and the MTI_Ben[3:0] is set appropriately based on the valid byte lines on the data buse (it is set to all 1's in this example). The write data is placed on the MTI_Data bus along with the MTI_Cmdvalid signal. The MAC110 Core asserted the M110_TxAck strobe for one clock, indicating the successful completion of the transaction and transfer of the write data.

4.1.2 Single Read transaction with Normal completion

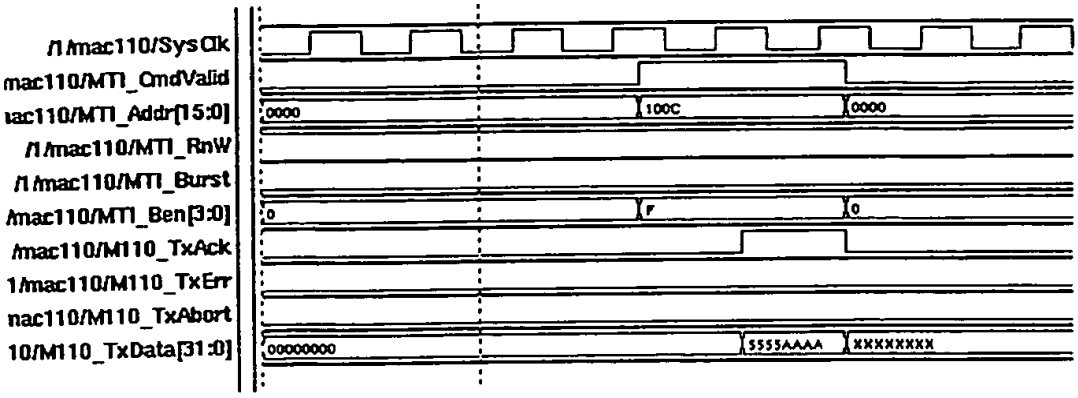


Figure 4-2: Single Read transaction with normal completion

Figure 4-2 shows a timing diagram in which the Application wants to do a single read transaction to address 16'h100C (Status Read). The Application asserts the MTI_CmdVallid signal and places the address on the MTI_Addr bus. The MTI_RnW is asserted high, indicating it as a read transaction, the MTI_Burst is deasserted, and the all the byte enables all set to 1's. The MAC110 Core asserts the M110_TxAck strobe for one clock, indicating the successful completion of the transaction and places the read data on the M110_TxData bus. This transaction is used by the Application to read the transmit status after the successful transmission of the frame or on seeing an abort for the previous write transaction.

4.1.3 Single Write transaction with Retry completion

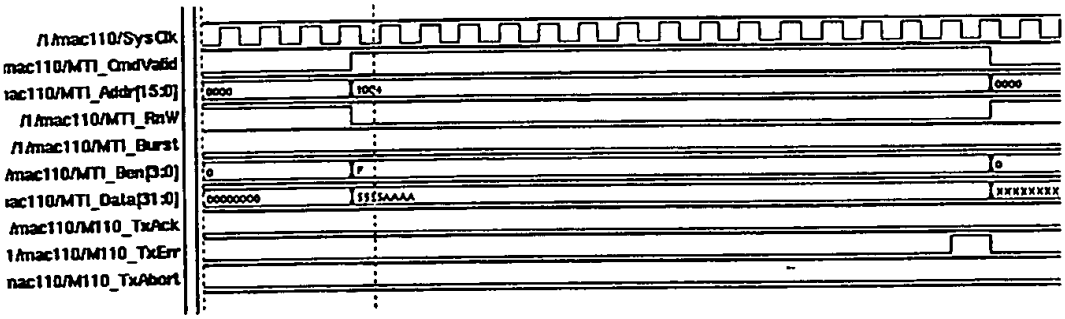


Figure 4-3: Single Write transaction with retry completion

Figure 4-3 shows a timing diagram in which the Application wants to do a single write transaction to address 16'h1004 (middle word write). The Application asserts the MTI_CmdValid signal and places the address on the MTI_Addr bus. The MTI_RnW is asserted low, indicating it as a write transaction, the MTI_Burst is deasserted and the byte-enables MTI_Ben[3:0] are set appropriately based on the valid byte lines on the MTI_Data bus (In this case all the byte-enables are set to 1's). The write data is placed on the MTI_Data bus along with the MTI_Cmdvalid signal. If the buffers become available with in the 15 clocks, the MAC110 core will assert the M110_TxAck and completes the transaction with data transfer. As the buffers in the MAC110 core are full, it waits for 15 clocks before issuing the M110_TxErr signal, indicating the Application to retry the transaction. The Application has to deassert the MTI_CmdValid signal and has to retry the transaction as soon as possible. The retry for the write transactions is possible when the Application is transferring either the middle words or the last word of the transmit frame.

4.1.4 Single Read transaction with retry completion

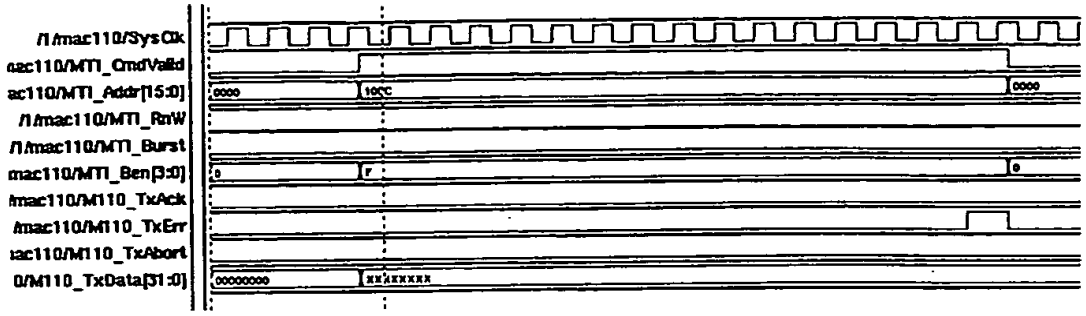


Figure 4-4: Single Read transaction with retry completion

Figure 4-4 shows a timing diagram in which the Application wants to do a single read transaction to address 16'h100C (Status Read). The Application asserts the MTI_CmdVallid signal and places the address on the MTI_Addr bus. The MTI_RnW is asserted high, indicating it as a read transaction, the MTI_Burst is deasserted, and the all the byte enables all set to 1's. The MAC110 Core tries to return the Status data with in with in 15 clocks, but if the status data is not available with in 15 clocks, the MAC110 core asserts the M110_TxErr signal indicating the Application to retry the transaction at a later time. This is the case when the application after transferring the frame data to the MAC110 core is trying to read the transmit status, but the MAC110 core may still be transmitting the frame (Pad/FCS fields). Hence the status is not available and issues retry to the status read transaction from the Application.

4.1.5 Single Write transaction with abort completion

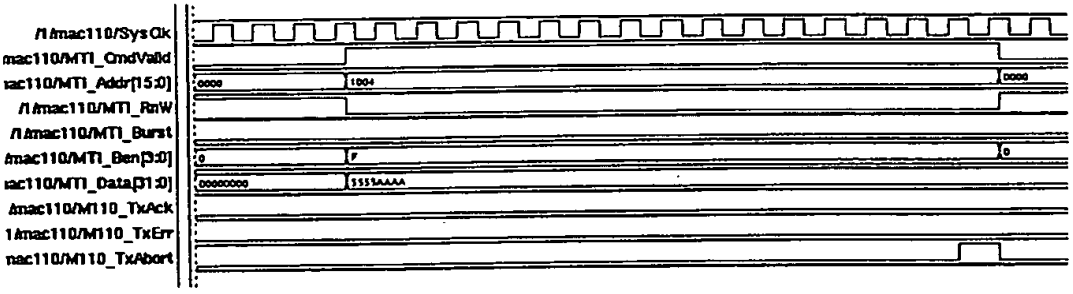


Figure 4-5: Single Write transaction with abort completion

Figure 4-5 shows a timing diagram in which the Application wants to do a single write transaction to address 16'h1004. The Application asserts the MTI_CmdVallid signal and places the address on the MTI_Addr bus. The MTI_RnW is asserted low, indicating it as a write transaction, the MTI_Burst is deasserted and the all the byte-enables MTI_Ben[3:0] are set appropriately based on the valid byte lines on the MTI_Data bus (In this case all the byte-enables are set to 1's). The write data is placed on the MTI_Data bus along with the MTI_Cmdvalid signal. If a regular collision or abort condition (late collision, under-run, excessive deferral, or excessive collisions) happens during the current frame's transmission, the MAC110 Core indicates the condition to the Application by aborting the current write transaction. The application has to deassert the MTI_CmdValid signal on observing the MAC110_Abort signal from the MAC110 Core. The Application has to do a status read transaction to get the status word from the MAC110 core about the cause of abort condition. If the status indicates that a collision happened, the application has to restart the frame transmission by transferring the first word of the frame.

4.2 Transmit Protocol

The Application initiates a new frame transmission by doing a write transaction on the transmit interface of the MAC110 Core with Address equal to ***0x1000***. The MAC110 Core accepts the data transfer by asserting the M110_TxAck signal. The MAC110 Core at this point will initiate a new frame transmission sequence on the MAC Block's transmit interface to indicate the MAC to start a new frame transmission.

Once the first write transfer is completed, the Application will (can continue the burst to) do multiple writes (single/burst) transactions to the MAC110 Core with the Address equal to ***0x1004***. The write transactions to this address indicate the continuation of the current frame transmission. Depending on the buffer status in the MAC110 Core, the MAC110 Core either accepts the transfers or retries the transfers. At any point, when there is an Abort to one of the write transaction, the Application should assume that the transmission of the frame on the Ethernet cable failed, and the Application must perform a read transaction with address equal to ***0x100C***, to get the Transmit status vector.

Once all except the last but one word have been transferred to the MAC110 Core, the Application has to do a last word write transaction to the MAC110 Core with Address equal to ***0x1008***. The MAC110 Core interprets a write transaction to this address as the last word transaction and the MAC110 Core will try to complete the transaction on the MAC transmit interface. Depending on the buffer status in the MAC110 Core, the MAC110 Core accepts the transaction or retries the transaction.

If not all the byte lines are valid on any of the *write* transfer, the application should set the appropriate byte-enables bit (MTI_Ben[3:0]) to logic '1'. Setting the byte enable bit to '1' indicates that the corresponding byte line is valid. The MAC110 core uses the data bytes that are valid (as indicated by the byte enables) and ignores the invalid byte lines.

Once the last write transaction is completed, the application has to read the Transmit status by doing a read transaction to the MAC110 Core with an Address equal to the ***0x100C***. The MAC110 Core returns the transmit status after the frame is completely transferred onto the Ethernet cable. If the MAC is still transmitting the current frame, the MAC110 Core will issue a retry to the read transaction. The read transaction is completed only after the packet is completely transferred onto the Ethernet cable and the MAC block returns the packet status.

If the packet status indicates that the current packet needs to be retried, the Application has to restart the transmission of the frame by doing a write transaction to Address ***0x1000***. The Application has to continue the rest of transactions until either the frame is successfully transmitted onto the Ethernet Cable or the frame is aborted because of an error condition. The MAC110 Core indicates the abort/retry condition on the Ethernet cable by aborting to any of the pending write transaction. The Application has to prematurely end the rest of the write transactions on an abort condition and read the Transmit packet status.

Figure 4-6 shows the sequence of flow and different transactions that are used to transfer a frame from Application to MAC110 Core. Table 4-1 shows the bit fields of the transmit status returned by the MAC110 Core.

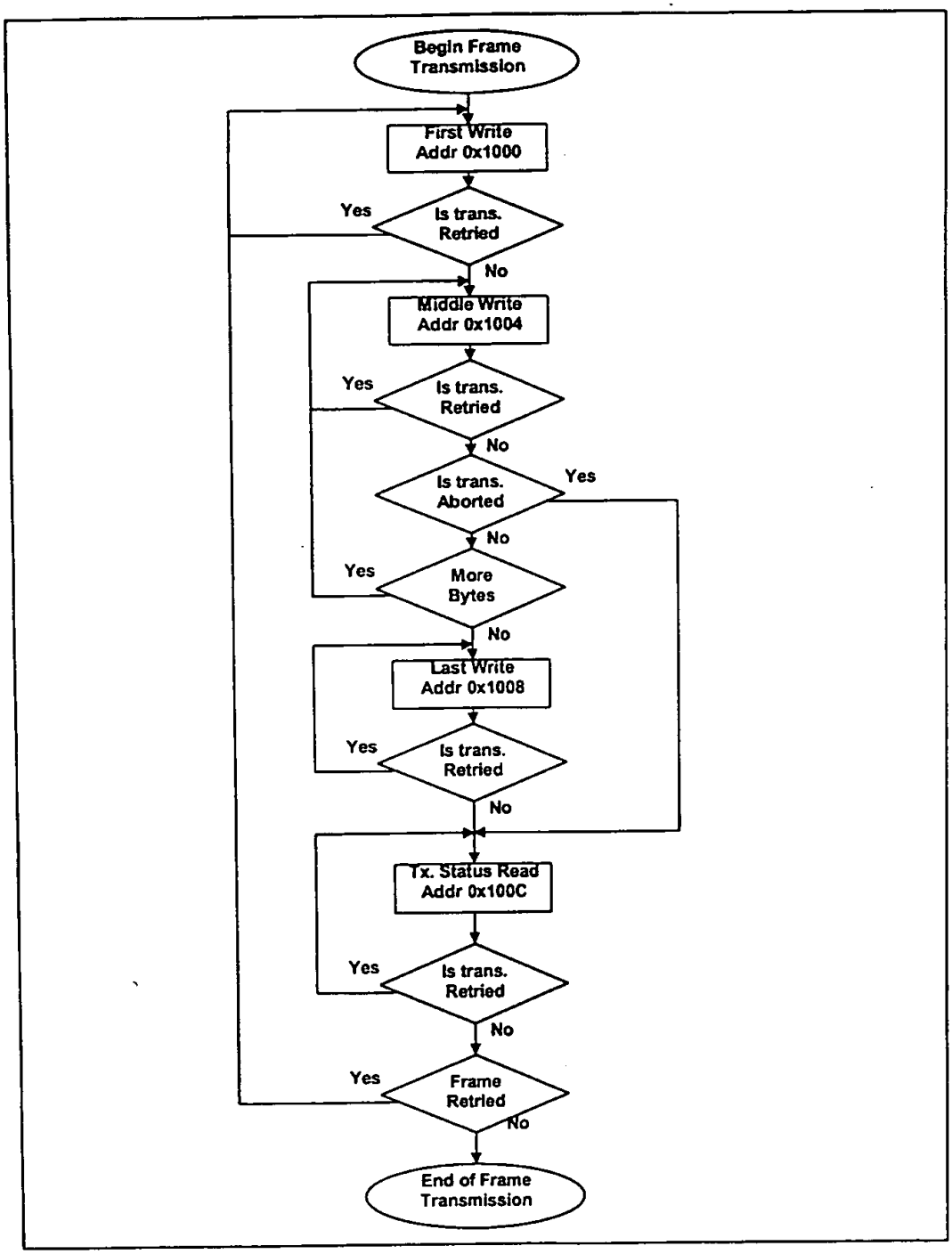


Figure 4-6: Flow chart for Transmit frame operation on the Transmit interface.

Field	Description
0	**Frame Aborted** When set, it indicates that the transmission of the current frame has been aborted by the MAC110 Core because of the following conditions. Jabber Timeout No Carrier Loss of Carrier Excessive Deferral Late Collision Retry Count exceeds the attemptLimit. Data under run If this bit is reset, it indicates that the current frame was successfully transmitted onto the MII Interface.
1	**Jabber Timeout** When set, indicates that the MAC110's transmitter has been active for an abnormally long time (twice the Ethernet maxFrameLength size).
2	**No Carrier** When set, indicates that the carrier signal form the transceiver was not present during transmission. This bit is valid only when the MAC110 core is operating in half-duplex mode.
3	**Loss of Carrier** When set, indicates that the loss of carrier occurred during the frame's transmission (i.e., the CRS input was inactive for one or more bit times when the frame is being transmitted). This is valid only for the frames transmitted and when the MAC is operating in half duplex mode.
4	**Excessive Deferral** When set, indicates that the transmission has ended because of excessive deferral of over 24,288 bit times during the transmission, if the defer bit is set high in the control register. This bit is valid only when the MAC110 core is operating in half-duplex mode.
5	**Late Collision** When set, indicates that the frame transmission was aborted due to collision occurring after the collision window of 64 bytes. Not valid if under run error is set. This bit is valid only when the MAC110 core is operating in half-duplex mode.
6	**Excessive Collisions** When set, indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the Disable Retry bit is set, this bit is set after the first collision and the transmission of the frame will be aborted. This bit is valid only when the MAC110 core is operating in half-duplex mode.
7	**Under Run** When set, indicates that the transmitter aborted the message because of the data under run. When the MAC110 Core doesn't have the data during the middle of frame's transmission, the Under Run bit is set.
8	**Deferred** When set, indicates that the transmitter had to defer while ready to transmit a frame because the carrier was asserted. This bit is valid only when the MAC110 core is operating in half-duplex mode.
9	**Late Collision Observed** When set, indicates that the MAC observed a late collision (collision after 64 bytes into transmission of frame), but retransmitted the frame in the next retransmission attempt. This is set when the Late Collision Control bit is set. This bit is valid only when the MAC110 core is operating in half-duplex mode.
13-10	**Collision Count** The 4-bit counter indicates the number of collisions that occurred before the frame was transmitted. Not valid when the Excessive Collisions bit is set. This bit is valid only when the MAC110 core is operating in half-duplex mode.
14	**Heart Beat Fail** This is effective only in 10-Mb/s operation mode and the Serial port is selected. When set, this bit indicates a heartbeat collision check failure (the transceiver failed to return a collision pulse as a check after the transmission). This bit is not valid if underflow error bit is set.
30-15	**Reserved for Future use.**

Field	Description
31	**Packet Retry** When set, it indicates that the current transmit packet has to be retried because of a collision on the bus. The Application has to restart the transmission of the frame (packet) when this bit is set to '1'. When the bit is reset, it indicates that the transmission of the current transmit packet is completed. The successful/unsuccessful completion of the frame's transmission is indicated by the bit 0.

Table 4-1: Bit fields of the Transmit Packet Status from the MAC110 Core

4.3 Receive Interface Transaction

On the receive interface, the transaction is always initiated by the MAC110 Core. The MAC110 Core initiates the transaction by asserting the M110_CmdValid signal and placing the transaction address on the M110_Addr[15:0] bus. The M110_RnW is always asserted, as the MAC110 core does only write transactions to Application on the receive interface. If the MAC110 Core wants to do a burst transaction, the M110_Burst will be asserted along with the M110_CmdValid signal. The data is placed on the M110_Data[31:0] bus. In response to this, the application has to assert the MRI_Ack signal with in four clocks to accept the data from the MAC110 core. If Application doesn't assert the MRI_Ack signal with in four clocks, the MAC110 *can* ignore the rest of the incoming frame from the Ethernet and sets the MissedFrame bit in the receive status. The flushing of the frame and the setting of the MissedFrame bit is based on the buffer status inside the MAC110 Core. The application can assert the MRI_Err signal for any of the transactions either to request the MAC110 core to retry the transaction or to stop the burst transaction. If the Application asserts the MRI_Err signal for any of the transactions coming from the MAC110 Core, the core *can* flush the data in the buffers, ignore the incoming packet and set the MissedFrame bit in the receive status. If the application asserts the MRI_Abort signal (to abort the transaction), the MAC110 core *will* terminate the current ongoing transaction, flushes the buffers in the core, flushes the incoming packet and sets the MissedFrame bit in the receive status. A detailed discussion on the receive protocol is discussed later in the chapter. The following sub-sections show timing diagrams to illustrate the various transactions on the Receive Interface.

4.3.1 Single Write with Normal completion

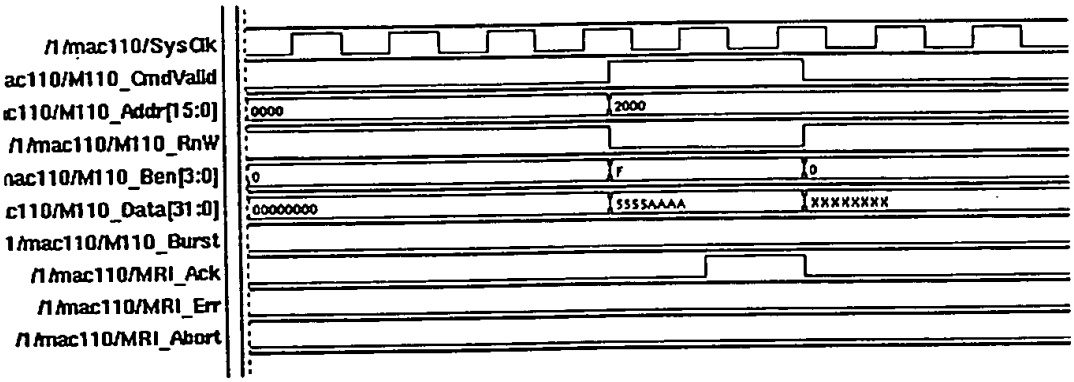


Figure 4-7: Single Write with normal completion

Figure 4-7 shows a timing diagram in which the MAC110 Core wants to do a single write transaction to address 16'h2000. The MAC110 Core asserts the M110_CmdVallid signal and places the address on the M110_Addr bus. The M110_RnW is asserted low, indicating it as a write transaction, the M110_Burst is dē-asserted and the all the byte enables all set to 1's. The write data is placed on the M110_Data bus along with the M110_Cmdvalid signal. The Application asserted the MRI_Ack strobe for one clock, indicating the successful completion of the transaction and transfer of the write data. The MAC110 Core uses this type of write transactions to transfer the frame information and transmit status word. The Application has to assert the MRI_Ack with in four clocks. If the Application delays the assertion of the MRI_Ack beyond 4 clocks, the MAC110 core *can* ignore the reset of the incoming receive frame and set the MissedFrame bit in the receive status.

4.3.2 Single Write with Retry completion

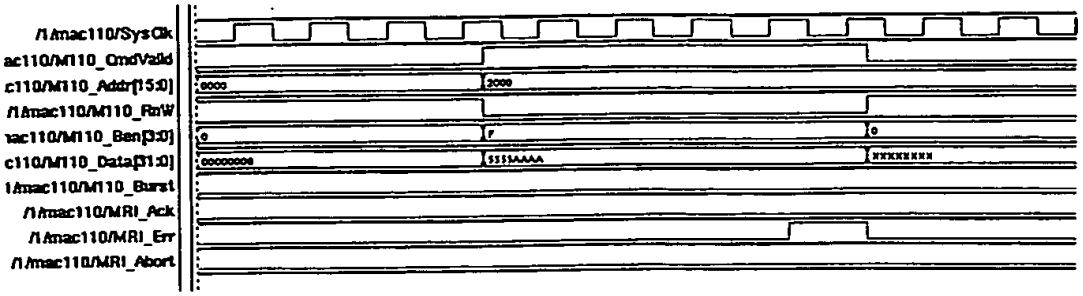


Figure 4-8: Single Write with Retry completion

Figure 4-8 shows a timing diagram in which the MAC110 Core wants to do a single write transaction to address 16'h2000. The MAC110 Core asserts the M110_CmdVallid signal and places the address on the M110_Addr bus. The M110_RnW is asserted low, indicating it as a write transaction, the M110_Burst is deasserted and the all the byte enables all set to 1's. The write data is placed on the M110_Data bus along with the M110_Cmdvalid signal. The Application asserted the MRI_Err signal for one clock indicating that the application cannot accept data at this point. The MAC110 core *can* ignore the rest of the incoming receive frame and sets the MissedFrame bit in the receive status. The flushing of the incoming frame and setting of the MissedFrame bit for a retry acknowledge from application is based on the MAC110 Core's current FIFO conditions and the incoming data rate. The MAC110 Core will try its best to handle the retry condition with the limited amount of buffer it has. If the FIFO gets full and the application issues retry, then the MAC110 Core flushes the FIFO and the incoming frame and sets the MissedFrame bit. The Application cannot assert the MRI_Err signal when the MAC110 is doing a write to address 16'h200C (Status write).

4.4 Receive Protocol

The MAC110 Core initiates a transmission of the newly received frame by doing a write transaction on the Receive Interface with an Address equal to ***0x2000***. The Application accepts the transaction, if the buffers in the Application are empty, by asserting the MRI_Ack signal.

Once the first write transaction is completed, the MAC110 Core will (can also continue the burst to) do multiple write (single/burst) transactions on the Receive Interface with the Address equal to ***0x2004***. The write transactions to this address indicate the continuation of the current receive frame transmission. Depending on the buffer status in the Application, the Application accepts the transaction or retries the transaction.

Once all except the last but one word (or part of the Word) has been transferred to the Application, the MAC110 Core will do a last word write transaction to the Application with address equal to ***0x2008***. The Application has to interpret a write transaction to this address as the last word transaction of the current received frame. Depending on the buffer status in the Application, the Application accepts the transaction or retries the transaction.

In case of the number bytes to be transferred are less than 4, the MAC110 Core will assert the byte-enables appropriately to identify the correct byte lines on the M110_Data bus in the last write transaction. The Application should use only the valid bytes in the last write transaction. (For all other transfers, the MAC110 core asserts all the byte enables).

Once the last write transaction is completed, the MAC110 Core transfers the receive status by doing a write transaction to the Application with an address equal to the ***0x200C***.

Except for the status transaction, the Application has to acknowledge the data write transactions with in four clocks. As the MAC110 Core has the limited amount of buffer to keep the incoming receive packet, the MAC110 Core expects that the Application will acknowledge the data write transaction as soon as possible. If the Application doesn't acknowledge the data transfer (inserts more than 4 wait states) with in 4 clocks, the MAC110 Core *can* drop the current receive frame by flushing the receive buffers in the core and not transferring the remaining frame to the application. The Receive status indicates the dropped frame condition (MissedFrame) and the Application should increment the missed frame counter appropriately (if it is maintaining one).

The MAC110 Core checks the incoming receive packet status from the MAC block and based on the control signals from the MCS block, it evaluates the Packet Filter status. For example if the Pass Bad Frames bit is reset in the in the MCS control register and an error frame is received, it will reset the Packet Filter status in the status word that is passed to the Application. The MAC110 Core uses the Receive All, Pass Bad Frames, Pass Control Frames, Disable Broadcast Frames control signals from MCS and generates the Packet Filter status.

Figure 4-9 shows the sequence of flow and different transactions that are used to transfer a frame from MAC110 to the Application. Table 4-2 shows the bit fields of the receive status provided by the MAC110 Core.

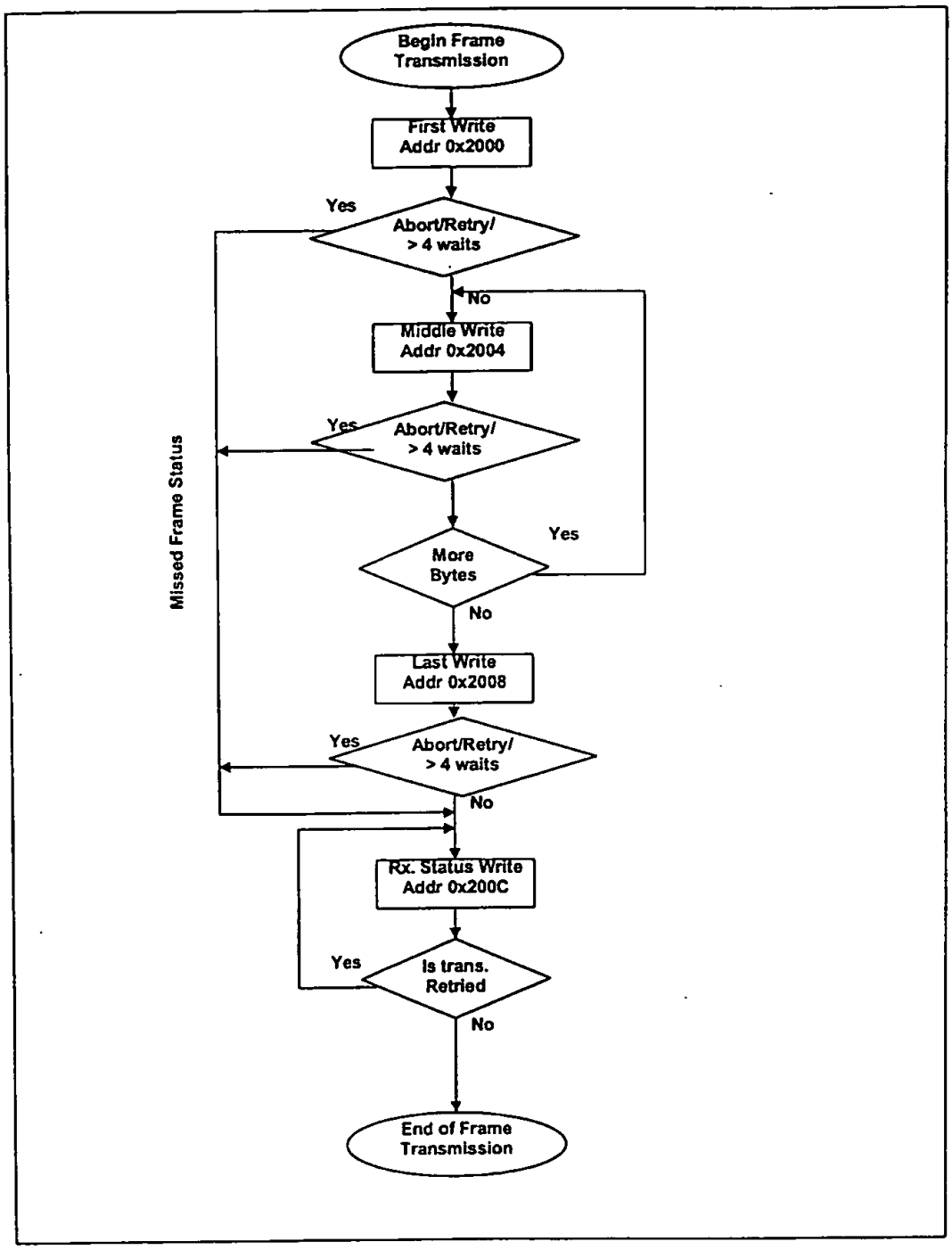


Figure 4-9: Flow chart for Receive frame operation on the MAC110-Recieve.

Field	Description
13-0	**Frame Length** Indicates the length in bytes, of the received frame, including pad (if applicable) the FCS field when the Automatic Pad Stripping bit is reset, else indicates the length with out pad and/or FCS field.
14	**Watchdog Timeout**
15	**Runt Frame** When set, indicates that this frame was damaged by a collision or premature termination before the collision window passed.
16	**Frame too Long** When set, indicates that the frame length exceeds the maximum Ethernet specified size of 1518 bytes. Frame too long is only a frame length indication and does not cause any frame truncation.
17	**Collision Seen** When set, indicates that the frame was damaged by a collision that occurred after the 64 bytes following the start of frame delimiter (SFD). This is a late collision.
18	**Frame Type** When set, indicates that the frame is an Ethernet-type frame (frame length field is greater than 1500 bytes). When clear, indicates the frame is an IEEE 802.3 frame. This bit is not valid for runt frames of less than 14 bytes.
19	**MII Error** When set, indicates that the MII_Rxer was asserted during the reception of this frame.
20	**Dribbling Bit** When set, indicates that the frame contained a non-integer multiple of 8 bits. This bit is not valid if either collision seen or runt frame bits are set. If set, and the CRC error is reset, then the packet is valid.
21	**CRC Error** When set, indicates that a cyclic redundancy check (CRC) error occurred on the received frame. This bit is also set when the MII_Rxer pin is asserted during the reception of a receive packet even though the CRC may be correct.
22	**One-Level VLAN Frame** When set, the current reception is tagged with a VLAN1 ID. The thirteenth and fourteenth bytes at the frame are compared to the one-level VLAN tag register. This bit is set if there is a non-zero match.
23	**Two-Level VLAN Frame** When set, the current reception is tagged with a VLAN2 ID. The thirteenth and fourteenth bytes at the frame are compared to the two-level VLAN tag register. This bit is set if there is a non-zero match.
24	**Length Error** When set, indicates that for the current frame the Length value is inconsistent with the total number of bytes received in the current frame. This is valid when the Frame Type is set to '0' (802.3 Frame).
25	**Control Frame** When set, the current frame is decoded as a control frame. This bit is set only when the MAC is operating in the full-duplex mode.
26	**Unsupported Control Frame** When set, it indicates that the MAC observed an unsupported Control Frame. This is set when a control frame is received and the opcode field is unsupported, or the length is not equal to minFrameSize (64 bytes). This bit is set only when the MAC is operating in the full-duplex mode. If the Control Frame is set and the Unsupported Control Frame is reset, it indicates that the MAC block received a valid Control Frame (PAUSE Command) and the transmitter is currently paused.
27	**Multicast Frame** When set, it indicates that the Destination Address in the current frame is a multicast address, i.e., the first bit of the DA is 1.

Field	Description
28	**BroadCast Frame** When set, it indicates that the Destination Address in the current frame is all 1's, i.e., broadcast address.
29	**Filtering Fail** When set, it indicates that the Destination Address field in the current frame failed the Address filtering. This bit is reset when it passes the address filtering.
30	**Packet Filter** When set, it indicates that the current frame passed the packet filter that is implemented in the MAC110 Core. The packet filter is based on the control signals from the MAC Control Register and the packet status from the MAC receive engine. When reset, it indicates that the current frame failed the packet filter. The Application can use this bit to decide whether to keep the packet in the memory or flush the packet from the memory/FIFO. Table 5-3 shows the Packet Filter output for various combinations of the control signals and the status signals.
31	**Missed Frame** This bit is set when the Application violates the wait clock latency protocol. If the Application doesn't assert the data acknowledge in 4 clocks for any of the write data transactions from MAC110 core or the application retries any of the write transactions, the MAC110 core cam drop the rest of the frame and transfers the Packet Status to the Application. If the application asserts the Abort signal for any of the write transactions, the MAC110 core will drop the frame. This bit is reset when the frame is received normally by the Application with out any latency/error violations.

Table 4-2: Bit fields of the Receive Packet Status from the MAC110 Core.

WE CLAIM:

1. A computer program product comprising computer readable program code for designing a controller, the computer readable program code comprising:

first program part for implementing an application interface;

second program part for implementing a physical layer interface; and

third program part for implementing a system-level function.

2. The computer program product of claim 1, wherein the first program part and the second program part also implement a link layer core.

3. The computer program product of claim 2, wherein the first program part and the second program part implement a link layer core which is compliant with an IEEE 802.3 standard.

4. The computer program product of claim 1, wherein the first program part implements an application interface comprising a virtual component interface ("VCI").

5. The computer program product of claim 1, wherein the second program part implements a physical layer interface comprising a media independent interface ("MII").

6. The computer program product of claim 1, wherein the third program part implements a system-level function comprising at least one of a reduced media independent interface ("MII") protocol, a serial MII interface protocol, a virtual component interface protocol, preamble generation and removal, thirty-two bit CRC generation and checking, insertion and stripping of padding bytes on transmission and reception, address filtering, a configurable counter, control of an MII compatible PHY, Carrier Sense Multiple Access Collision Detection ("CSMA/CD") protocol, and a test environment.

7. A computer program product comprising computer readable program code for designing a controller, the computer readable program code comprising:

first program part for substantially implementing a medium access control sublayer protocol; and

second program part for implementing a system-level function.

8. The computer program product of claim 7, wherein the first program part is for substantially implementing a data link layer protocol.

9. A method of designing a controller using a software package, the method comprising:

utilizing the software package which comprises an application interface functionality, a physical layer interface functionality, and a system-level function functionality;

incorporating the application interface functionality into a controller design;

incorporating the physical layer interface functionality into the controller design; and

incorporating the system-level function functionality into the controller design.

10. A method of designing a controller using a software package, the method comprising:

utilizing the software package which comprises a medium access control sublayer functionality and a system-level function functionality;

incorporating the medium access control sublayer functionality into a controller design; and

incorporating the system-level function functionality into the controller design.

11. The method of claim 10, wherein the software comprises a functionality for substantially all of a link layer core, and further comprising incorporating the functionality for substantially all of a link layer core into the controller design.

12. A design tool for designing a controller, the design tool comprising:

a medium access control sublayer emulator which substantially emulates a medium access control sublayer protocol; and

a system-level function emulator.

13. The design tool of claim 12, further comprising a data link layer emulator which substantially emulates a data link layer protocol.

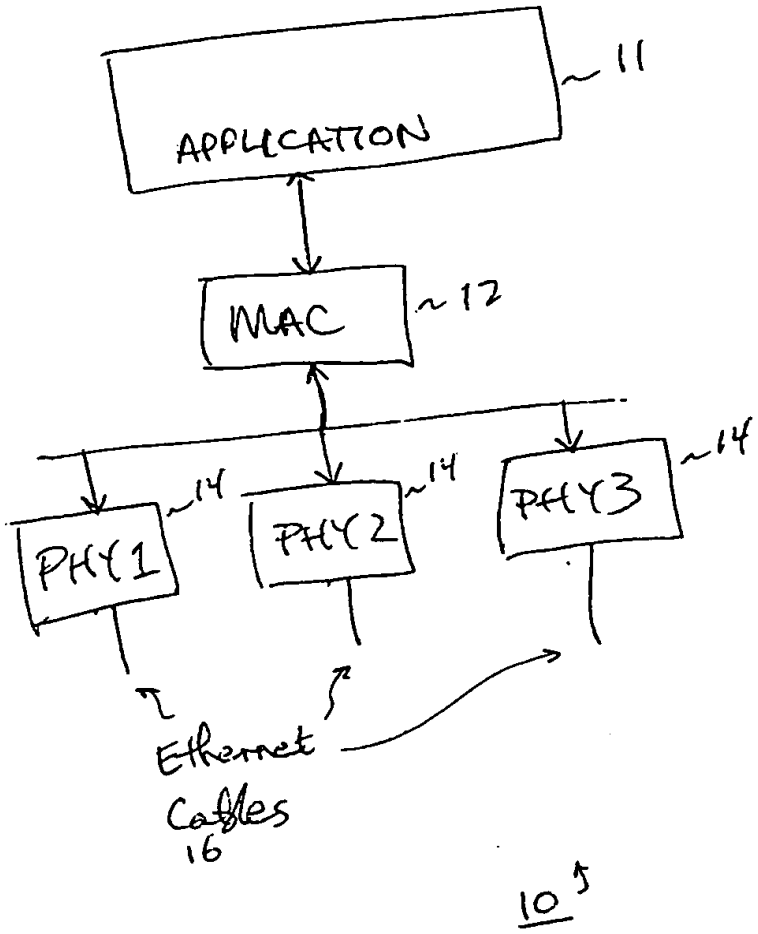

APPLICATION
11
MAC
12
PHY1
14
PHY2
14
PHY3
14
Ethernet
Cables
16
10

FIG. 1

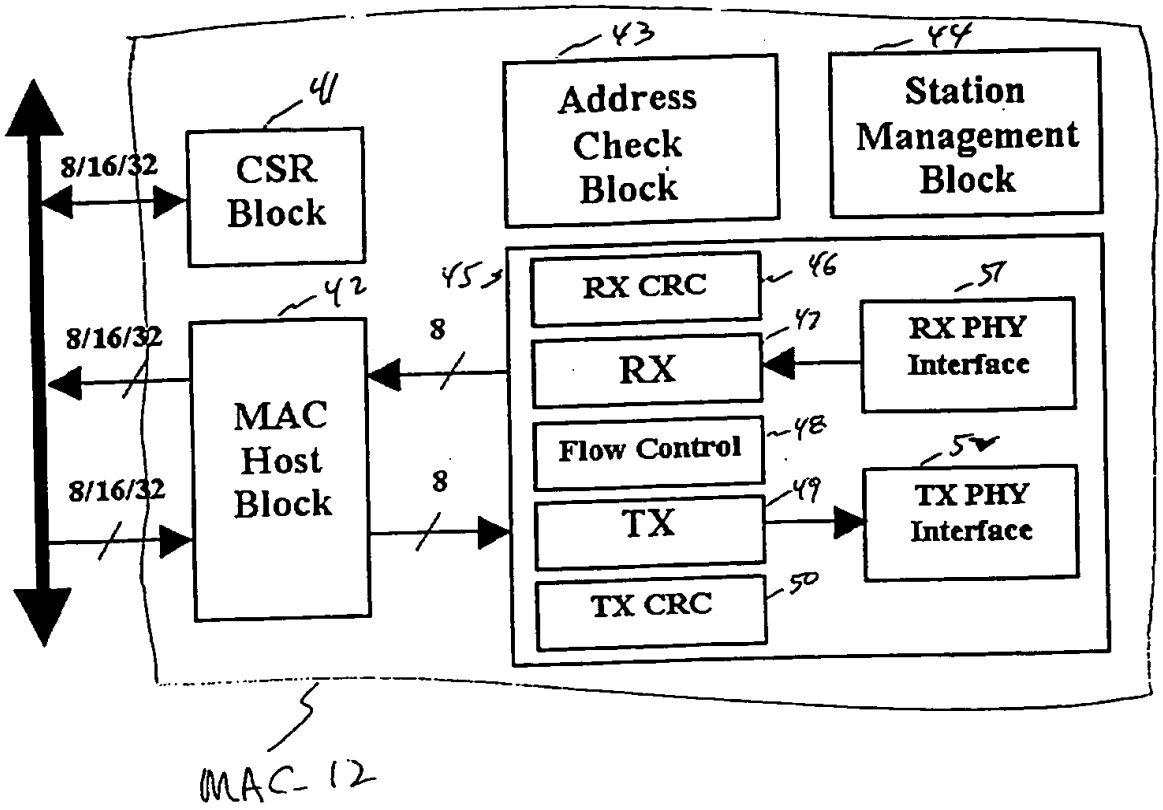

41
43
44
Address Check Block
Station Management Block
CSR Block
8/16/32
45
RX CRC
46
47
RX PHY Interface
RX
42
8/16/32
8
MAC Host Block
Flow Control
48
51
8/16/32
8
TX
49
TX PHY Interface
TX CRC
50
MAC-12

FIG. 2

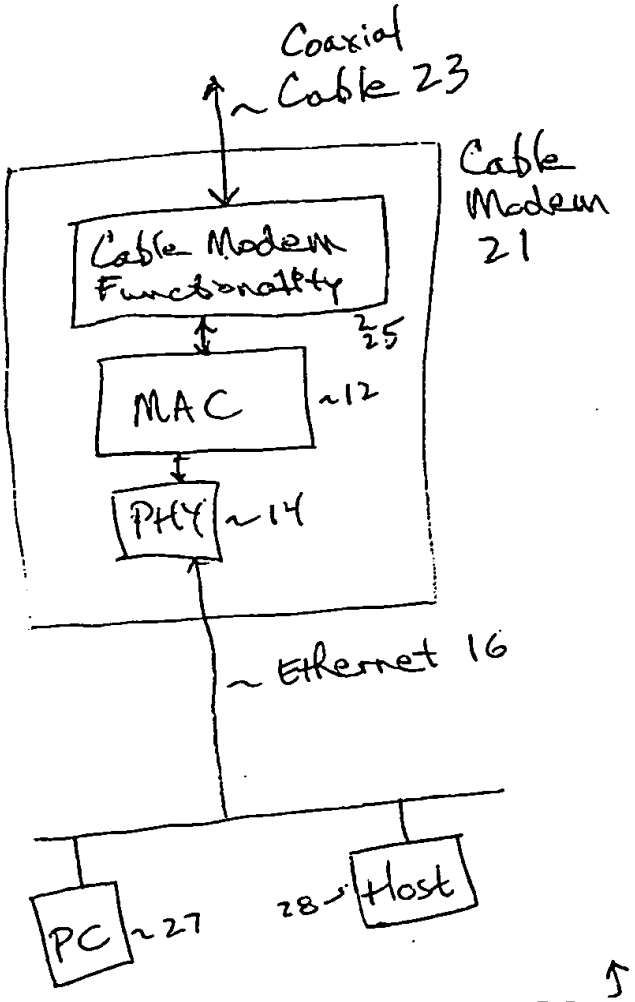
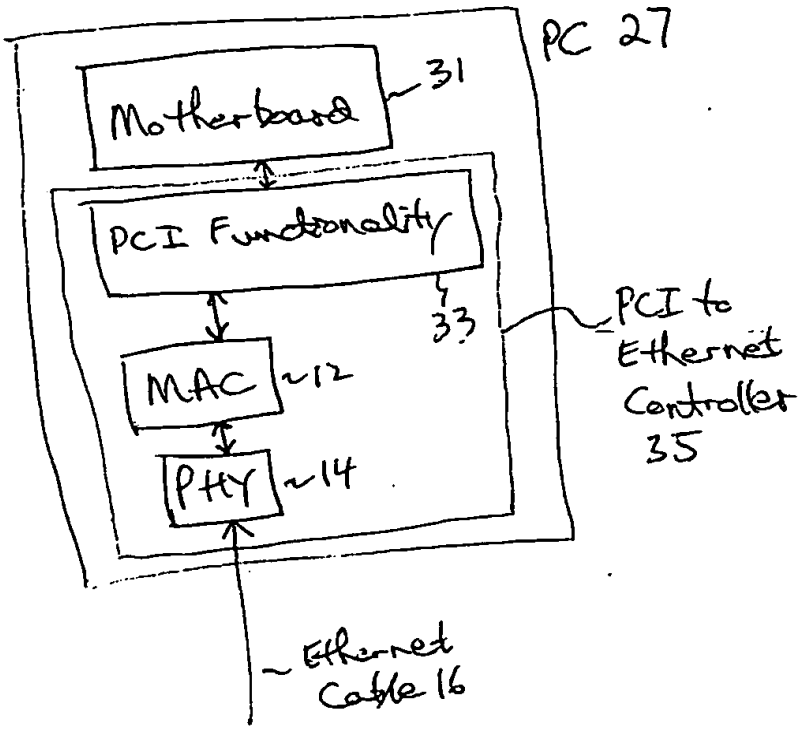

Coaxial
Cable 23
Cable
Modem
21
Cable Modem
Functionality
25
MAC
12
PHY
14
Ethernet 16
PC
27
28
Host
20

FIG. 3



30

FIG. 4A

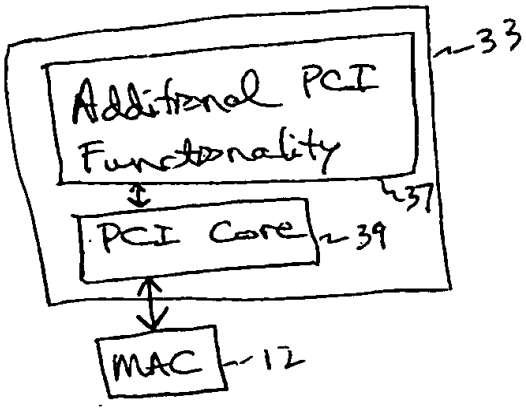


FIG. 4B